WHAT IS CLAIMED IS:

1. A method for operating a computer, comprising:

storing in a computer memory a plurality of pseudocode instructions, at least some of said pseudocode instructions comprising a plurality of machine code instructions;

for each of a plurality of tasks or jobs to be performed by the computer, automatically creating a respective virtual thread of execution context data including (a) a memory location of a next one of said pseudocode instructions to be executed in carrying out the respective task or job and (b) the values of any local variables required for carrying out the respective task or job, a plurality of said tasks or jobs each entailing execution of a respective one of said pseudocode instructions comprising a plurality of machine language instructions;

processing each of said tasks or jobs in a respective series of time slices or processing slots under the control of the respective virtual thread; and

in every context switch between different virtual threads, undertaking such context switch only after completed execution of a currently executing one of said pseudocode instructions.

2. The method defined in claim 1 wherein each of the virtual threads is part of a respective linked list of virtual threads, each of the virtual threads further including a pointer to a next virtual thread in the respective linked list, further comprising, for every context switch between different virtual threads, consulting the pointer of a currently

executing virtual thread to determine an identity of a next virtual thread to be executed.

3. The method defined in claim 2 wherein said respective linked list is one of a plurality of linked lists of said virtual threads, one of said linked lists being a list of idle virtual threads, another of said linked lists being a list of active virtual threads, an additional one of said linked lists being a list of queued virtual threads, further comprising periodically moving at least one virtual thread from said list of queued virtual threads to said list of active virtual threads.

4. The method defined in claim 3 wherein the moving of a virtual thread from said list of queued virtual threads to said list of active virtual threads includes:

setting a mutex to lock said list of queued virtual threads;

subsequently modifying pointers in (i) the moved virtual thread, (ii) at least one virtual thread originally in said list of active virtual threads, and (iii) at least one virtual thread remaining in said list of queued virtual threads; and

thereafter resetting or releasing the mutex to enable access to said list of queued virtual threads.

5. The method defined in claim 1 wherein each of said virtual threads additionally includes a mutex, further comprising:

setting the mutex of a selected one of said virtual threads;

subsequently modifying data in said selected one of said virtual threads; and

thereafter resetting or releasing the mutex to enable access to said selected one of said virtual threads.

6. The method defined in claim 5 wherein the setting of said mutex of said selected one of said virtual threads, the modifying of said data, and the resetting or releasing of said mutex of said selected one of said virtual threads are performed in response to a message from one other of said virtual threads.

7. The method defined in claim 5 wherein each of the virtual threads is part of a respective linked list of virtual threads, each of the virtual threads further including a pointer to a next virtual thread in the respective linked list, the modifying of said data including modifying a pointer of said selected one of said virtual threads.

8. The method defined in claim 1 wherein each of said virtual threads is assigned a message queue, further comprising entering a message in a message queue of a selected one of said virtual threads during execution of a task or job pursuant to another one of said virtual threads.

9. The method defined in claim 8 wherein said selected one of said virtual threads and said another one of said virtual threads correspond to respective tasks or jobs derived from different applications programs, whereby the entering of said message in the message queue of said selected one of said virtual threads implements

data transfer between said different applications programs.

10. The method defined in claim 8 wherein said selected one of said virtual threads and said another one of said virtual threads are proxy or interface threads on different computers, the entering of said message in said message queue including transmitting said message over a communications link between said computers.

11. The method defined in claim 1 wherein the creating of the virtual threads, the processing of said tasks or jobs in respective series of time slices or processing slots, and the undertaking of context switches all include the operating of the computer under an interpreter program.

12. The method defined in claim 11, further comprising running a plurality of instances of said interpreter program on the computer, each instance corresponding to a native thread, each native thread:

creating a respective set of virtual threads of execution context data;

processing each of a plurality of tasks or jobs in a respective series of time slices or processing slots under the control of the respective virtual thread; and

in every context switch between different virtual threads, undertaking such context switch only after completed execution of a currently executing one of said pseudocode instructions.

13. The method defined in claim 12, further comprising shifting a virtual thread from a first native thread having a heavier-than-average load to a second native thread having a lighter-than-average load.

14. The method defined in claim 13 wherein the shifting of a virtual thread includes:

determining an average load over all the native threads by summing thread load values for the native threads and dividing by the number of threads; and

for each of the native threads, comparing the respective thread load value with the average load to determine relative load.

15. The method defined in claim 1 wherein said virtual threads include a first proxy thread for communicating with a second proxy thread on another computer via a computer network link, the processing of a communication with said another computer including using standard network protocols under the control of said first proxy thread.

16. The method defined in claim 15 wherein each of said virtual threads, including said first proxy thread, is assigned a respective message queue, further comprising entering a message in a message queue of said first proxy thread to execute a data transfer to said another computer over said computer network link.

17. The method defined in claim 1 wherein a selected one of said virtual threads

is in an idle state, further comprising:

generating a message in response to an input from a source outside the computer;

inserting said message in a message queue for said selected one of said virtual threads;

changing said selected one of said virtual threads from said idle state to an active state;

after the inserting of said message in said message queue and the changing of the state of said selected one of said virtual threads, accessing said message queue to obtain said message during a time slice or processing slot assigned to said selected one of said virtual threads.

18. The method defined in claim 1 wherein each of said virtual threads additionally includes a thread priority, further comprising automatically consulting the thread priorities in a plurality of said virtual threads to determine relative priorities and varying a sequence of threads in accordance with the determined relative priorities.

19. The method defined in claim 1 wherein the tasks or jobs processed in respective series of time slices or processing slots under the control of the respective virtual threads include:

controlling objects imaged on a computer display, each of said objects constituting a separate task or job assigned a respective one of said virtual threads; and

monitoring actuation of keys on a computer keyboard, each of said keys constituting a separate task or job assigned a respective one of said virtual threads.

20. The method defined in claim 1 wherein said time slots or processing slots are measured by counting consecutively executed pseudocode instructions, further comprising, for each of a plurality of said time slices or processing slots, terminating the respective time slot or processing slot upon counting a predetermined number of consecutively executed pseudocode instructions.

21. A multi-tasking computer comprising:

a memory;

a display;

an input peripheral;

at least one processor operatively connected to said memory, said display, and said input peripheral, said processor having:

a compiler for converting operator-entered source code instructions into bytecode or pseudocode instructions, said compiler being operatively linked to said memory for enabling the storage of said bytecode or pseudocode instructions therein; and

an interpreter for executing said bytecode or pseudocode instructions,

said memory storing a first linked list of idle virtual threads, a second linked list of active virtual threads, and a third linked list of queued or waiting virtual threads, each of said threads including context or state data, a mutex and a pointer to a next thread in

the respective list, said interpreter being operatively connected to said input peripheral for recognizing an event generated by said input peripheral, said interpreter being operatively connected to said memory (a) for shifting at least one of said idle virtual threads from said first linked list to said third linked list, (b) for shifting queued or waiting virtual threads from said third linked list to said second linked list, (c) for executing instructions according to context and state data of different virtual threads in said second linked list in successive time slices or processing slots pursuant to a predetermined priority schedule, said interpreter being operatively connected to said display in part for modifying an object on said display in response to instructions specified by a respective active virtual thread in said second linked list.

22. The computer defined in claim 21 wherein:

said memory additionally stores a fourth linked list of native threads;

said interpreter is one of a plurality of instances of a common interpreter, each of said instances of said common interpreter corresponding to a respective one of said native threads;

said second linked list is one of a plurality of linked active-thread lists, each of said native threads being linked by a respective pointer to a respective one of said linked active-thread lists; and

said third linked list is one of a plurality of linked queued-thread lists, each of said native threads being linked by a respective pointer to a respective one of said linked queued-thread lists.

23. The computer defined in claim 22 wherein said active threads each includes a mutex for enabling locking of the respective thread by one native thread to prevent access to the respective thread by other native threads.

24. The method defined in claim 22 wherein said interpreter includes means shifting a virtual thread from a first native thread having a heavier-than-average load to a second native thread having a lighter-than-average load.

25. The computer defined in claim 21 wherein said list of idle virtual threads includes a plurality of threads assigned to respective keys of a keyboard for processing actuations of the respective keys.

26. The computer defined in claim 21 wherein said list of idle threads includes a plurality of threads assigned to respective objects in a display image for processing changes in appearance of the respective objects.

27. The computer defined in claim 21 wherein said interpreter includes a context switch module and a instruction counter, said context switch module being operatively connected to said memory and said instruction counter for effectuating a context switch from a currently executing active thread of said second linked list to a next active thread in said second linked list upon execution of a predetermined number of bytecode or pseudocode instructions pursuant to said currently executing active thread.

28. The computer defined in claim 21 wherein each of said virtual threads includes a memory location of a next instruction to execute in the respective thread, values of any local variables for the respective thread, and an execution priority for the respective thread.

29. The computer defined in claim 21 wherein said memory stores a plurality of message queues assigned to respective ones of said threads.

30. The computer defined in claim 21 wherein said memory stores at least one proxy or interface thread having an execution context for carrying out a communication with a remote computer via a communications link, said proxy or interface thread containing a memory address leading to a network protocol routine.

31. In a computer having an interpreter for executing a series of bytecode instructions each consisting of a multiplicity of machine code steps, a multitasking method comprising:

for each task of a plurality of tasks to be performed by the computer, using the interpreter to define a respective virtual thread;

during each time slice of a series of consecutive time slices, executing bytecode instructions of a respective current thread selected from among the virtual threads; and

executing a context switch from one of said virtual threads to another of said virtual threads only after execution of one of said bytecode instructions.

32. The method defined in claim 31 wherein each of said virtual threads is part of a respective linked list of virtual threads, each of the virtual threads further including a pointer to a next virtual thread in the respective linked list, further comprising, for every context switch between different virtual threads, consulting the pointer of a currently executing virtual thread to determine an identity of a next virtual thread to be executed.

33. The method defined in claim 32 wherein said respective linked list is one of a plurality of linked lists of said virtual threads, one of said linked lists being a list of idle virtual threads, another of said linked lists being a list of active virtual threads, an additional one of said linked lists being a list of queued virtual threads, further comprising periodically moving at least one virtual thread from said list of queued virtual threads to said list of active virtual threads.

34. The method defined in claim 33 wherein the moving of a virtual thread from said list of queued virtual threads to said list of active virtual threads includes:

setting a mutex to lock said list of queued virtual threads;

subsequently modifying pointers in (i) the moved virtual thread, (ii) at least one virtual thread originally in said list of active virtual threads, and (iii) at least one virtual thread remaining in said list of queued virtual threads; and

thereafter resetting or releasing the mutex to enable access to said list of queued virtual threads.

35. The method defined in claim 31 wherein each of said virtual threads additionally includes a mutex, further comprising:

setting the mutex of a selected one of said virtual threads;

subsequently modifying data in said selected one of said virtual threads; and

thereafter resetting or releasing the mutex to enable access to said selected one of said virtual threads.

36. The method defined in claim 35 wherein the setting of said mutex of said selected one of said virtual threads, the modifying of said data, and the resetting or releasing of said mutex of said selected one of said virtual threads are performed in response to a message from one other of said virtual threads.

37. The method defined in claim 31 wherein each of said virtual threads is assigned a message queue, further comprising entering a message in a message queue of a selected one of said virtual threads during execution of a task or job pursuant to another one of said virtual threads.

38. The method defined in claim 31 wherein said virtual threads include a first proxy thread for communicating with a second proxy thread on another computer via a communications link, further comprising processing bytecode instructions according to said first proxy thread for sending a message to said second proxy thread over said

communications link.

39. The method defined in claim 31 wherein each of said virtual threads additionally includes a thread priority, further comprising automatically consulting the thread priorities in a plurality of said virtual threads to determine relative priorities and varying a sequence of threads in accordance with the determined relative priorities.

40. The method defined in claim 31 wherein said time slots or processing slots are measured by counting consecutively executed pseudocode instructions, further comprising, for each of a plurality of said time slices or processing slots, terminating the respective time slot or processing slot upon counting a predetermined number of consecutively executed pseudocode instructions.

41. A multi-tasking computer comprising:

a memory storing state and context data of multiple threads or tasks;

an interpreter for executing a series of bytecode instructions each consisting of a multiplicity of machine code steps, the interpreter being programmed:

to define a respective virtual thread for each task to be performed by the computer;

to execute bytecode instructions of a respective current thread selected from among the virtual threads during each time slice of a series of consecutive time slices; and

to execute a context switch from one of said virtual threads to another of said virtual threads only after execution of one of said bytecode instructions.

42. The computer defined in claim 41 wherein each of said virtual threads is part of a respective linked list of virtual threads, each of the virtual threads further including a pointer to a next virtual thread in the respective linked list, said interpreter being further programmed to consult, for every context switch between different virtual threads, the pointer of a currently executing virtual thread to determine an identity of a next virtual thread to be executed.

43. The computer defined in claim 42 wherein said respective linked list is one of a plurality of linked lists of said virtual threads, one of said linked lists being a list of idle virtual threads, another of said linked lists being a list of active virtual threads, an additional one of said linked lists being a list of queued virtual threads, said interpreter being further programmed to periodically move at least one virtual thread from said list of queued virtual threads to said list of active virtual threads.

44. A computer method comprising:

running a timer of a computer to generate a series of time slices or processing slots;

compiling input user source code into byte- or pseudocode instructions each corresponding to a multiplicity of machine code instructions;

operating an interpreter of said computer to assign computing tasks to respective virtual threads, the assigning of said computing tasks to said virtual threads including identifying and storing state and context data for each of said computing tasks;

in each of said time slices, additionally operating said interpreter to execute selected ones of said byte- or pseudocode instructions pursuant to the state and context data of a current one of said virtual threads;

after the execution of each successive one of the selected byte- or pseudocode instructions and only after such execution, further operating said interpreter to check whether a predetermined interval has elapsed since a commencement of execution of instructions pursuant to said current one of said virtual threads; and

upon a determination of elapsing of said predetermined interval, operating said interpreter to perform a context switch.

45. The method set forth in claim 44 wherein the tasks assigned to respective ones of said virtual program threads include (a) controlling objects appearing in an image on a display screen, (b) monitoring operator input, (c) executing routines of applications programs, (d) running computer maintenance routines, (e) carrying out communications with remote computers via a computer network, and (f) calculating local variables.